Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Vladimír Bedecs

Automatické rozpoznávání gest myši za účelem ovládání aplikací

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Petr Homola

Studijní program: Informatika

2007

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 28. 5. 2007 Vladimír Bedecs

Obsah

Název práce: Automatické rozpoznávání gest myši za účelem ovládání aplikací

Autor: Vladimír Bedecs

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Petr Homola

e-mail vedoucího: Petr.Homola@mff.cuni.cz

Abstrakt: V předložené práci studujeme problematiku automatického rozpoznávání gest myši, za účelem ovládání aplikací. To zahrnuje vytvoření programu, který je schopný rozpoznávat předem definovaná gesta prováděná myší a umožnit pomocí nich ovládání aplikací. Program je napsán pro operační systém Windows. Práce obsahuje popis několika algoritmů pro rozpoznávání gest myši, které jsou v přiloženém programu Mouse Gestures použité. Také zkoumá techniku háků operačního systému a její využití pro ovládání aplikací. Důležitou kapitolou práce je vykonávání významu gesta, která pozůstává z návrhu a implementace struktur nesoucích význam gesta, aby vykonávání významu gesta myši bylo co nejefektivnější. V závěru práce je uvedeno několik rad a návodů na používání zhotovené aplikace.

Klíčová slova: myš, gesto, rozpoznávání, ovládání

Title: Automatic recognition of mouse gestures for application controlling

Author: Vladimír Bedecs

Department: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Petr Homola

Supervisor’s e-mail address: Petr.Homola@mff.cuni.cz

Abstract: In the present work we study problems of automatic recognition of mouse gestures for application controlling. This includes creation of program, which is able to recognize already defined gestures made by mouse and with them provides controlling of applications. The program is written or operating system Windows. This work includes description of some algorithms for recognition of mouse gestures, which are used in enclosed program Mouse Gestures. Also analyzes hook techniques of operating system and uses them for application controlling. An important chapter of this work is processing of gesture’s meaning, which includes gesture’s meaning structures implementation, so that the processing of gesture’s meaning can be most effective. In the last pages of this work can reader find some advices and tutorials for using made application.

Key words: mouse, gesture, recognition, controlling

Úvod

Každodenná práca s počítačom, predstavuje pre mnohých užívateľov sústavné opakovanie jednotlivých príkazov či operácií s operačným systémom, čo vedie ku zbytočným úkonom, ktoré by sa dali vykonať jednou klávesovou skratkou, alebo iným signálom. Táto skutočnosť vedie k úvahám, v akej podobe dávať signály pre aplikáciu ktorá ich spracuje a ako by malo samotné zjednodušenie práce, teda vykonávanie spomínaných rutinných príkazov, prebiehať?

Odhliadnuc od možnosti spúšťania príkazov klávesovými skratkami existuje na tieto otázky viacero odpovedí. Avšak jedna z atraktívnych možností je spracovávanie gest myši, ako signál, ktorý dáva impulz pre vykonávanie jednotlivých príkazov. Na zrealizovanie podobnej myšlienky je nutné vyvinúť algoritmy, ktoré budú schopné spracovať gestá myši a takisto aj mechanizmus, ktorý zabezpečí korektné ovládanie aplikácií. V jednotlivých kapitolách tejto práce je popísaná algoritmická stránka rozpoznávania gest myši, ďalej sú uvedené základné princípy mechanizmu operačného systému, ktorý umožňuje získavať informácie o užívateľových akciách a následne sa tieto informácie zužitkujú v predposlednej kapitole v podobe interpretácie týchto akcií pre ovládanie aplikácií. Posledná kapitola je venovaná stručným radám a návodom, ako používať priloženú aplikáciu Mouse Gestures, ktorá je implementáciou uvedených postupov a algoritmov. Obsah predloženého textu sa často krát odvoláva na samotnú aplikáciu.

Väčšina úryvkov zdrojového kódu je v jazyku C a demonštrujú volania API funkcií operačného systému Windows, keďže aplikácia Mouse Gestures, je určená práve pre túto platformu.

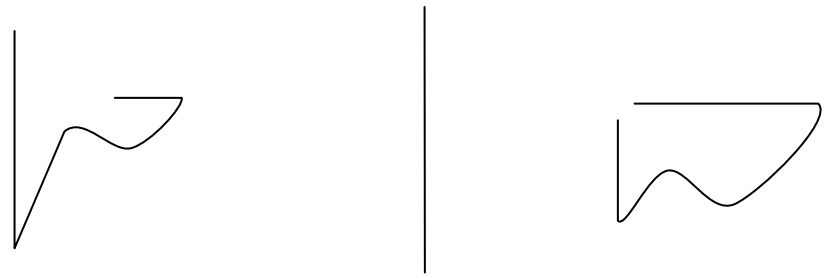
Kapitola 1

Algoritmy vyhodnocovania gest myši

1.1 Motivácia a popis problému

Keďže je takmer nemožné nakresliť s myšou v ruke opakovane trajektóriu, ktorá by sa zhodovala s predošlou v každom bode na monitore a tým ich označiť za rovnaké. Je nutné zaviesť algoritmy na rozpoznávanie, ktoré budú tolerovať menšie či väčšie odchýlky a označia za rovnaké aj trajektórie, ktoré sa nezhodujú v každom svojom bode.

Definícia: Za *tvarovo totožné*, resp. *(logicky) podobné trajektórie* sa budú v nasledujúcom texte považovať dve trajektórie, ktoré znázorňujú rovnaký, resp. (logicky) podobný tvar v rovine.



Obrázok č. 1 Dve totožné, resp. podobné trajektórie.

Existuje niekoľko prístupov, či spôsobov ako rozpoznať, resp. vyhodnotiť trajektóriu gesta myši. Podstata každého spôsobu vyhodnocovania však zostáva zachovaná a tou je snaha vyhodnotiť dve tvarovo totožné, resp. (logicky) podobné trajektórie za rovnaké.

Výstupom každej metódy (algoritmu) je jedna z možností:

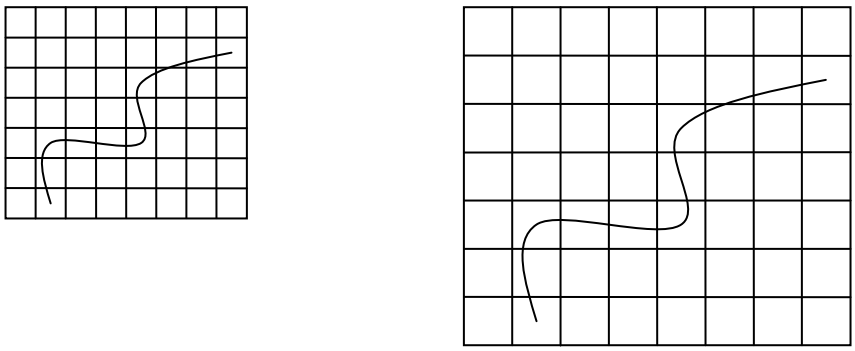
- **množina aktívnych oblastí** ktorými trajektória prechádza
- **postupnosť (identifikátorov) smerov** popisujúca priebeh trajektórie (ďalej len reťazec gesta).

1.2 O algoritmoch

Počet bodov tvoriacich trajektóriu gesta je závislý na rýchlosti pohybu s myšou, keďže poloha myši je zaznamenávaná len v určitých pravidelných intervaloch operačným systémom. Všetky algoritmy sú v istých situáciách pri vyhodnocovaní reťazca gesta silne závislé na hustote bodov v trajektórii. V nasledujúcich častiach textu sa vyskytnú jednotlivé algoritmy, či metódy zoradené podľa zložitosti a svojich schopností od najjednoduchších po komplexné vyhodnocovacie algoritmy. Každý z nich je parametrizovateľný a zvolená hodnota parametru do veľkej miery ovplyvňuje výsledok výpočtu, či dokonca aj jeho správnosť.

1.3 Štvorcová metóda

Štvorcový algoritmus je jednou z najjednoduchších metód kategorizácie ľubovoľnej trajektórie, a teda aj trajektórie myši. Rovina, v ktorej trajektória leží sa mriežkou rozdelí na rozmerovo totožné oblasti (štvoruholníky). Je to teda prechod od spojitého priestoru do diskrétneho. Rozmery jednotlivých oblastí (jemnosť mriežky) je nutné prispôsobiť rozmerom trajektórie, aby bol tvar trajektórie kategorizovaný rovnako aj pri rôznych zväčšeniach mierky trajektórie.



Obrázok č. 2 Prispôsobenie rozmerov mriežky gestu.

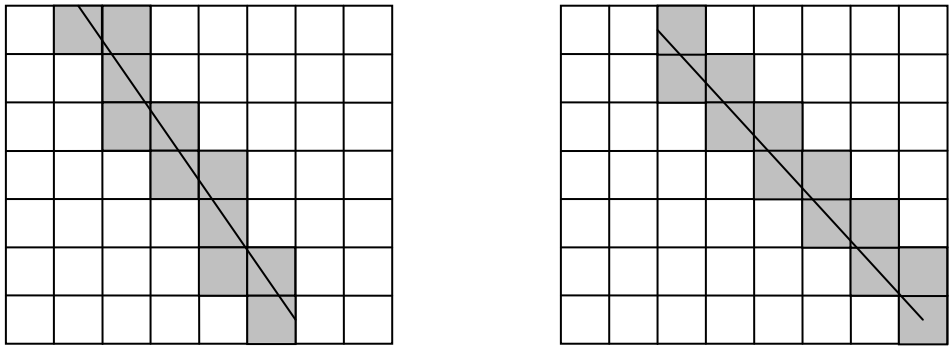
Táto metóda má niekoľko výhod a je veľmi ľahko implementovateľná. Jednou z výhod je, že rozdelenie spojitého priestoru diskrétnou mriežkou povoľuje odchýlky pri opätovnom interpretovaní (pri užívateľskom kreslení myšou) trajektórie. Prijateľnú odchýlku určuje jemnosť mriežky, čiže veľkosť jednotlivých oblastí, ktoré sa skúmajú pri rozpoznávaní trajektórie.

Priebeh výpočtu

Každá oblasť nadobúda pri vyhodnocovaní dva stavy: aktívna (do oblasti zasahuje trajektória) a neaktívna (v oblasti sa trajektória nevyskytuje). Výstupom štvorcovej metódy je množina aktívnych oblastí, ktorú dokáže skonštruovať v lineárnom čase vzhľadom na počet bodov v trajektórii.

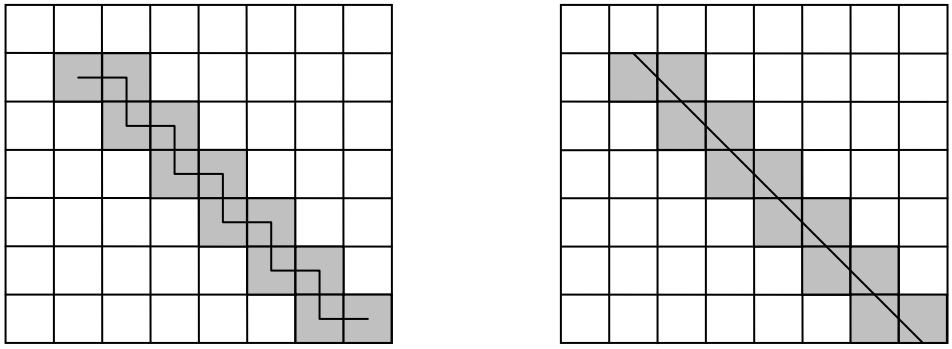
Závažnými nedostatkami tejto metódy je nízka úspešnosť rozpoznávania diagonálnych smerov a nutnosť zachovať presnú proporcionalitu trajektórie pri jej interpretácii. Tým sa tento algoritmus stáva prakticky nepoužiteľným pre rozpoznávanie gest myši, v prípade zložitejších tvarov gest. Preto sa mu v texte ďalej nevenuje pozornosť.

Demonštrácia chýb algoritmu



Obrázok č. 3 Nesprávne vyhodnotenie totožných trajektórii za odlišné.

Demonštrácia nesprávneho vyhodnotenia rôznych (logicky nepodobných) trajektórií štvorcovou metódou za rovnaké.



Obrázok č. 4 Nesprávne vyhodnotenie rôznych trajektórií za totoňé.

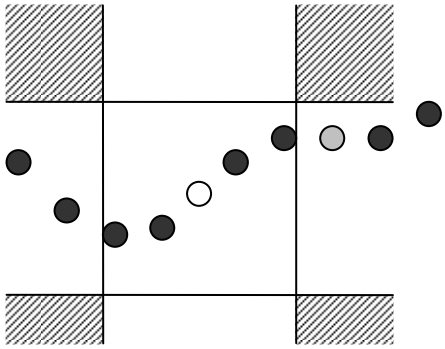
1.4 Štvorsmerový algoritmus

Štvorsmerový algoritmus rieši niektoré nedostatky štvorcovej metódy. Výstupom algoritmu je postupnosť smerov, ktoré trajektória nadobúda vo svojom priebehu. Ako jeho názov napovedá, je schopný rozpoznávať štyri možné smery priebehu gesta. Algoritmus prechádza všetky zaznamenané body trajektórie od počiatku až do konca a popri tom generuje v lineárnom čase reťazec gesta.

Pri skúmaní trajektórie operuje s troma typmi bodov: aktívnym, pasívnym a skúmaným. Aktívny bod je bod trajektórie, vzhľadom na ktorý sa skúma poloha ďalšieho (skúmaného) bodu ležiaceho v geste.

Priebeh výpočtu

Parametrom štvorsmerového algoritmu je kritická vzdialenosť skúmaného bodu od aktívneho. Ako náhle je skúmaný bod vzdialený od aktívneho aspoň na kritickú vzdialenosť, určí sa a pridá sa do reťazca gesta príslušná poloha skúmaného vzhľadom na aktívny, skúmaný bod prechádza do aktívneho stavu a z aktívneho bodu sa stáva pasívny. Ak skúmaný nie je vzdialený od aktívneho aspoň na kritickú vzdialenosť, tak sa zo skúmaného bodu stáva pasívny a za skúmaný sa označí nasledujúci bod ležiaci v trajektórii gesta.



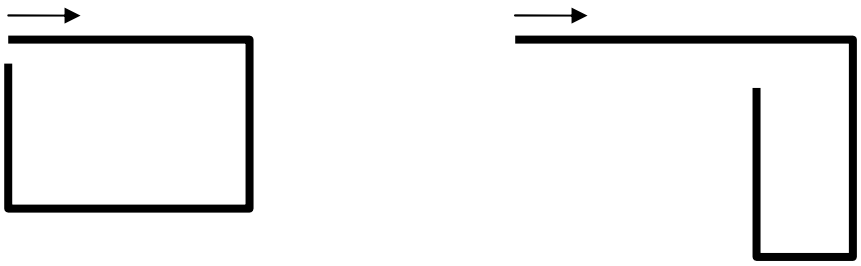
Obrázok č. 5 Hľadanie nového aktívneho bodu trajektórie.

Po spracovaní všetkých bodov v trajektórii týmto spôsobom, získame reťazec gesta (postupnosť smerov) zložený so štyroch možných smerov: hore, dole, doprava, doľava. Obmedzenie veľkosti množiny smerov len na štyri je nutné, pretože vyhodnotenie polohy skúmaného bodu, ktorý prekročil kritickú hranicu, vzhľadom na aktívny, ako diagonálnej (skúmaný bod leží vo vyšrafovanej časti obrázku č. 5),

sa vyskytuje zriedka. Táto skutočnosť vyplýva z faktu, že vyšrafovaná plocha na obrázku má s oblasťou obsahujúcou aktívny bod (aktívna oblasť) len štyri spoločné hraničné body a tým pádom má skúmaný bod vyššiu pravdepodobnosť prekročiť hranicu aktívnej oblasti s nevyšrafovanými oblasťami. Každá nevyšrafovaná časť predstavuje svojou polohou vzhľadom na oblasť obsahujúcu aktívny bod jeden zo štyroch smerov.

Demonštrácia chýb algoritmu

Na obrázku č. 6 vľavo aj vpravo vyhodnotí štvorsmerový algoritmus v oboch prípadoch trajektórie ako totožné gestá, aj keď nie sú (logicky) podobné, keďže dĺžky jednotlivých hrán nakreslených obrazcov (dĺžka smeru) si nezodpovedajú. Reťazce oboch gest sú: doprava, dole, doľava, hore. Štvorsmerový algoritmus teda zanedbáva dĺžky smerov, čo však môže byť v niektorých prípadoch jeho výhodou.



Obrázok č. 6 Nesprávne vyhodnotenie trajektórií za totožné.

1.5 Osemsmerový algoritmus

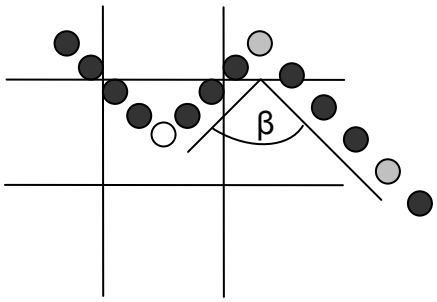
Osemsmerový algoritmus pre rozpoznávanie gest myši je funkčnou nadstavbou štvorsmerového algoritmu, ktorou je spoľahlivá kategorizácia ďalších štyroch diagonálnych smerov. Je takisto aj základom pre oblúkový algoritmus rozpoznávania gest myši. Výpočtovo a implementačne je však náročnejší, ale stále zachováva lineárnu zložitosť vzhľadom na počet bodov v trajektórii gesta. Táto metóda je založená na meraní uhlov, ktoré zvierajú priamky tvorené bodmi trajektórie. Obraznejší názov algoritmu znie „hľadanie rohov“. Výstupom algoritmu je opäť postupnosť smerov, ktoré nadobúda trajektória gesta vo svojom priebehu.

Táto metóda operuje so štyrmi typmi bodov trajektórie: aktívny, skúmaný, pomocný a pasívny. Parametrami algoritmu sú takzvaný „zlomový uhol“, určujúci minimálny uhol medzi dvoma priamkami, prechádzajúce aktívnym, skúmaným

a pomocným bodom, aby bol ich prienik považovaný za významný („roh“), kritická vzdialenosť s totožným významom ako v štvorsmerovom algoritme a premenná určujúca posun pomocného bodu od skúmaného v trajektórii.

Priebeh výpočtu

Na začiatku výpočtu sa označí prvý bod trajektórie ako aktívny a hľadá sa prvý výskyt prieniku priamok tvorenými aktívnym, skúmaným a pomocným bodom trajektórie, ktoré zvierajú menší uhol (viď. vzorec (1.1) [1] str. 557) ako zlomový. Prienik priamok sa nachádza v skúmanom bode. Ak teda splňuje predošlú podmienku a je zároveň vzdialený od aktívneho bodu aspoň na kritickú vzdialenosť, vyhodnotí a zaznamená sa do reťazca smerov gesta jeho poloha vzhľadom na aktívny bod a označí sa za aktívny. Výpočet prebieha analogicky, až kým algoritmus nenarazí na posledný bod trajektórie.



Obrázok č. 7 Meranie uhlu medzi aktívnym, skúmaným a pomocným bodom.

$$\cos\beta = \frac{u_0 v_0 + u_1 v_1}{\sqrt{(u_0^2 + u_1^2)(v_0^2 + v_1^2)}} \qquad (1.1.)$$

Je zrejmé, že týmto spôsobom je zaručené korektné rozpoznanie všetkých smerov priebehu trajektórie vrátane tých diagonálnych (vpravo hore, vpravo dole, vľavo hore, vľavo dole). Voľba hodnoty posunu pomocného bodu od aktuálne skúmaného je závislá na hustote zaznamenaných bodov v trajektórii. Väčšinou sa však volí (vzhľadom na hardwarové schopnosti a priemernú rýchlosť kreslenia gesta myši rukou užívateľa) z intervalu <3, 10>.

Osemsmerový algoritmus je jednoducho rozšíriteľný dokonca aj na základnú kategorizáciu proporcií gesta (dĺžok jednotlivých smerov). Stačí tým pádom iba zmerať vzdialenosť medzi aktívnym bodom a novo nájdeným „rohom trajektórie“ a vhodne ju kategorizovať v rámci reťazca gesta.

Demonštračný kód osemsmerového algoritmu

```
{ Vytvori osemsmerovym algoritmom z trajektorie string gesta }
function GetStr(ang:angle; dist,con: integer; traj:Traject):string
var
    ret : string;
    i : integer;
    active, current, pom : Point;
    way : EightWays;

begin
    ret := "";
    active := traj[1];

    for i := 2 to traj.length() – con do
        begin
            current := traj[i];
            pom := traj[i + con];
            {
              Ak priamky zvieraju mensi alebo rovnaky uhol ako
              zlomovy, alebo pomocny bod je posledny v trajektorii
            }
            if (Angle(active, current, pom) <= ang) or
               (i + con = traj.length()) then
                begin
                    {
                      Poloha skumeneho bodu vzhladom na aktivny.
                      Je to NOTHING, ak skumany bod je blizsie ako
                      v kritickej vzdialnosti.
                    }
                    way := GetDirection(active, current, dist);
                    if way <> NOTHING then
                        begin
                            if ret.last() <> way.char() then
                                ret += way.char();
                            active := current;
                        end;
                end;
        end;
    GetStr := ret;
end;
```

1.6 Oblúkový algoritmus

Je opäť nadstavbou schopností osemsmerového algoritmu o rozpoznávanie oblúkov v nakreslenej trajektórii. Algoritmus priamo využíva osemsmerový algoritmus pre vygenerovanie reťazca gesta. Následne skúma tento reťazec a vlastnosti trajektórie pre vygenerovanie reťazca gesta obsahujúci (identifikátory oblúkov) oblúky. V ďalšom texte sa predpokladá, že osemsmerový algoritmus, ktorý využíva oblúkový, nemá implementovanú schopnosť rozlišovania dĺžok smerov gesta. Aj keď mierna modifikácia oblúkového algoritmu, by bola schopná úspešne využívať aj túto jeho funkčnosť.

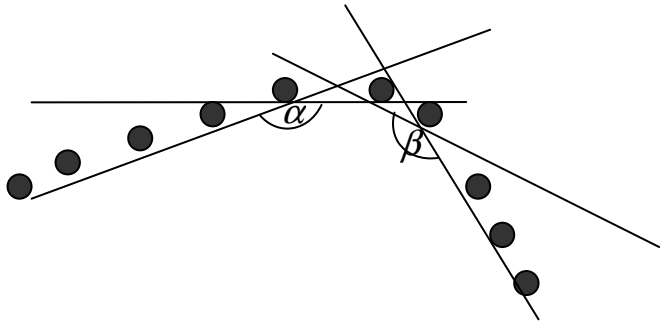
Oblúkový algoritmus má lineárnu zložitosť O(n), kde n je počet zaznamenaných bodov v trajektórii. Základná myšlienka algoritmu je veľmi jednoduchá. Vo vygenerovanom reťazci gesta osemsmerovým algoritmom hľadá postupnosť smerov (smerové slová), ktoré by mohli tvoriť kružnicový oblúk. Ak zistí, že skúmané smerové slovo naozaj oblúk tvorí, tak do reťazca gesta vloží identifikátor príslušného kružnicového výseku. Ak skúmané smerové slovo netvorí časť kružnice, tak do reťazca gesta zaznamená pôvodné smerové slovo.

Priebeh výpočtu

Pri výpočte algoritmu sa teda vyhľadávajú kľúčové smerové slová v reťazci gesta vygenerovaným osemsmerovým algoritmom, ktoré sa prípadne prepisujú pomocou prepisovacích pravidiel oblúkového algoritmu do reťazca gesta obsahujúci oblúky. Na obrázku č. 8 sú pred šípkou znázornené tvary trajektórie, ktoré sú prepísané po splnení podmienky pre oblúk (podmienka hladkosti), prepisovacím pravidlom do reťazca gesta ako (identifikátor oblúka) oblúk. Parameter tohto algoritmu nazvime kritická hodnota.

Obrázok č. 8 Prepisovanie úsekov trajektórie do oblúkovej formy.

Definícia: Nech body b1, b2, b3, b4, b5 sú susednými bodmi trajektórie a priamky P1 vedená bodmi b1, b2 a priamka P2 vedená bodmi b2, b3 zvierajú uhol α a priamky P3 vedená bodmi b3, b4 a priamka P4 vedená bodmi b4, b5 zvierajú uhol β . Potom hovoríme, že uhly α a β sú *blízke*.



Obrázok č. 9 Blízke uhly.

Definícia: Nech uhly α a β sú blízke, potom hovoríme, že úsek trajektórie b1...b5 je *hladký*, ak $|\alpha - \beta| \leq k$, kde k je kritická hodnota algoritmu.

Podmienka hladkosti je splnená pre istú časť trajektórie, ak sú všetky jej úseky (zložené s bodov u1...u5) hladké. Táto podmienka je jediné kritérium, ktoré musia časti trajektórie tvoriace po rozpoznaní osemsmerovým algoritmom kľúčové smerové slová oblúkového algoritmu splniť, aby boli prepísané na oblúky.

Ukážka smerových prepisovacích pravidiel

hore, doprava => oblúk hore doprava

hore, šikmo hore doprava, doprava => oblúk hore doprava

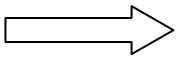
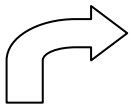
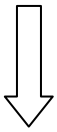
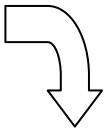
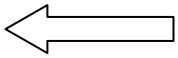
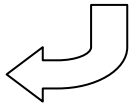
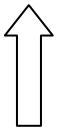
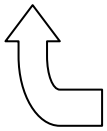
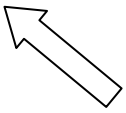
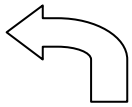
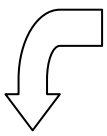
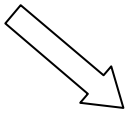
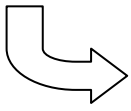
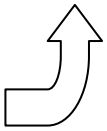
doprava, dole => oblúk dole doprava

doprava, šikmo dole doprava, dole => oblúk dole doprava

šikmo hore doľava, doľava => oblúk hore doľava

1.7 Klasifikácia smerov pohybu myši aplikáciou Mouse Gestures

Aplikácia Mouse Gestures implementuje tri z uvedených algoritmov na rozpoznávanie gest myši. Konkrétne: „Štvorsmerový algoritmus“, „Osemsmerový algoritmus“ (bez vyhodnocovania dĺžok hrán) a „Oblúkový algoritmus“. Po nakreslení gesta myši vygeneruje pomocou aktuálne zvoleného algoritmu reťazec gesta (postupnosť smerov pohybu). Reťazec je zložený z písmen vyjadrujúcich jednotlivé smery podľa pravidla:

Smer gesta	Reťazec	Smer gesta	Reťazec
	R		I
	D		F
	L		G
	U		H
	W		E
	X		A
	Z		B
	Y		C

Tabuľka č. 1 Klasifikácia smerov v Mouse Gestures.

Kapitola 2

Systémové háky správ OS Windows

2.1 Motivácia a popis problému

Bežnými spôsobmi je takmer nemožné, aby aplikácia spustená v operačnom systéme Windows, dostávala informácie o ostatných bežiacich aplikáciách, či nepretržite kontrolovala zariadenia ako sú myš, alebo klávesnica, teda aj vo chvíľach, k nie je aktívna (nemá fokus).

Klasická aplikácia dostáva do svojej fronty správ len tie správy, ktoré sú určené niektorému z jej okien alebo správy od operačného systému, ktoré majú charakter celosystémovej udalosti, oznámenia o vypínaní OS, odhlásenie aktuálne prihláseného užívateľa, alebo zmeny v systémových nastaveniach.

Ak teda chceme byť (naša aplikácia chce byť) schopný zisťovať aj správy patriace oknám iných aplikácií, ktoré neboli primárne určené aj pre našu, musíme nasadiť neštandardný mechanizmus, ktorým sú bezpochyby systémové háky správ. Tie nám umožnia monitorovať v každom okamžiku polohu myši, užívateľský klávesový vstup, či spúšťanie a ukončovanie celých aplikácií, prípadne len otváranie a zatváranie ich okien. V každom okamžiku znamená aj vtedy, ak okno našej aplikácie nie je aktívne. Tento postup je schopný zachytiť dokonca ľubovoľnú správu, ktorá bola odoslaná alebo prijatá aplikáciou, bez ohľadu na to, či je systémová, alebo ju priamo svojou činnosťou vyvolal užívateľ. V nasledujúcom texte budú popísané základné skupiny systémových hákov, ich schopnosti a rozhrania.

Musím poznamenať, že v tejto kapitole mi bola nesmierne nápomocná publikácia Radka Chalupu [3], z ktorej som čerpal množstvo poznatkov.

2.2 Systémové háky správ

Aby sme boli schopný reagovať na ľubovoľné správy určené oknu našej aplikácie, musí ich prijať do svojej fronty správ, čo zabezpečuje operačný systém, odkiaľ ju dobre známym spôsobom (API funkciou *GetMessage* v cykle) odosiela procedúre príslušného okna na ďalšie spracovanie. Samozrejme drvivú väčšinu správ aplikácia

ignoruje a z mohutnej množiny správ Windows reaguje len na tie, ktoré ju zaujímajú. Otázkou teda zostáva, ako nám pomôžu systémové háky správ aby sme uvedeným postupom boli schopný prijímať aj správy určené pre okná iných aplikácií.

Háky správ predstavujú mechanizmus, ktorý odošle správu okrem jej adresáta (okno pre ktoré je určená) aj takzvanej procedúre háku, pred tým alebo potom, čo je správa doručená na pôvodné miesto určenia.

Základné rozdelenie typov hákov

- Háky monitorujúce našu (jednu) aplikáciu spolu so všetkými jej oknami. Nachádzajú využitie v situáciách, keď chceme na jednu konkrétnu udalosť reagovať v rámci celej našej aplikácii (vo všetkých jej oknách) tým istým spôsobom, teda jednou spoločnou procedúrou. Tým pádom odpadá nutnosť písať v procedúrach okien opakovane ten istý kód pre obsluhu spomínanej udalosti.
- Háky monitorujúce všetky spustené aplikácie, prípadne aktívne procesy so všetkými ich oknami. Používajú sa v situáciách, keď potrebujeme získavať informácie o udalostiach týkajúce sa buď inej aplikácie alebo celého systému.

Medzi týmito dvoma typmi hákov je však zásadný rozdiel, čo sa týka umiestenia procedúr hákov. Zatiaľ, kým prvá skupina hákov (háky monitorujúce našu (jednu) aplikáciu) sú umiestnené priamo v zdrojovom kóde vlastnej aplikácie, háky druhého typu (háky monitorujúce všetky spustené aplikácie), musia byť kvôli architektúre operačného systému Windows umiestnené v samostatne dynamicky linkovanej knižnici, teda nie v zdrojovom kóde našej aplikácie.

Hákov správ existuje niekoľko typov líšiacich sa v množine správ, ktoré sú schopné zachytávať, v časoch v ktorých sa procedúra háku volá a v schopnostiach zachytiť, prípadne modifikovať získanú správu. Každému háku zodpovedá príslušný formát procedúry háku.

Množiny správ zachytávajúce jednotlivé háky

Množiny správ, ktoré sú jednotlivé háky schopné zachytiť nie sú disjunktné, a teda je nutné si pred nasadením akéhokoľvek háku v aplikácii poriadne rozmyslieť, ktorá množina čo najpresnejšie popisuje tú, ktorú potrebujeme mať k dispozícii.

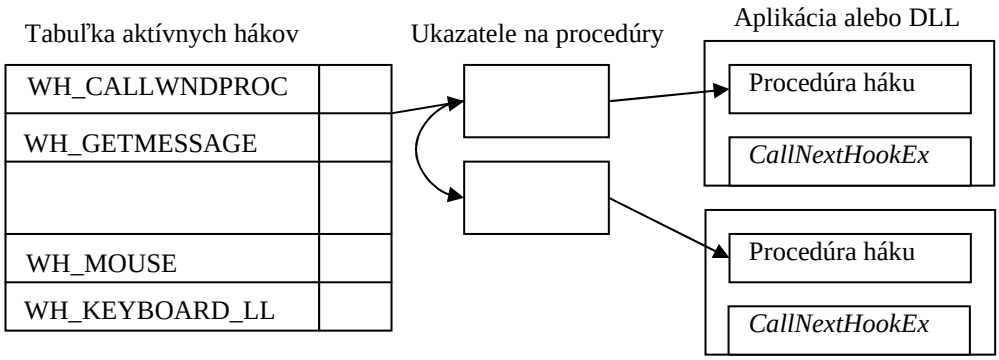
Napríklad množina správ háku typu *WH_MOUSE* je podmnožinou správ háku typu *WH_MOUSE_LL*, ktorá je však opäť podmnožinou správ, ktorú je schopný zachytiť hák typu *WH_GETMESSAGE*, ktorého procedúra sa zavolá vždy, keď je pomocou známych funkcií *GetMessage* alebo *PeekMessage* vybratá správa z fronty správ aplikácie. Spomínaný hák typu *WH_MOUSE*, ako už názov sám vypovedá, zachytáva úzku skupinu správ týkajúcich sa myši. Takisto existuje aj hák, zachytávajúci správy klávesnice (hák typu *WH_KEYBOARD*). Za zmienku stojí aj hák typu *WH_CALLWNDPROC*, ktorého procedúra prijíma informácie o zaslaní ľubovoľnej správy pomocou funkcie *SendMessage*.

2.3 Systémové háky správ a priebeh volaní

Ako bolo už spomenuté, v okamžiku implementácie jedného zo systémových hákov správ je nutné, aby bol umiestený (jeho procedúra) v samostatnej dynamicky linkovanej knižnici. Keďže kód systémového háku sa musí nachádzať v adresovom priestore aplikácie, ktorej správy chceme „odpočúvať“, musíme ju nejakým spôsobom „prinútiť“, aby si do svojho adresového priestoru pomocou funkcie *LoadLibrary* načítala kód danej dynamickej knižnice obsahujúcej procedúru háku. Neexistuje žiaden iný spôsob, vzhľadom na zabezpečenie stability systému, aby mala jedna aplikácia prístup do pamäťového priestoru iných aplikácií. Aby už bežiaca aplikácia, ale aj tá, ktorá sa práve len spúšťa, zavolala funkciu *LoadLibrary* na knižnicu s našim pripraveným hákom, je nutné zavolať príslušné API funkcie nastavujúce háky správ.

Nemenej dôležité je aj správne odhlásenie systémových hákov v chvíli, keď uznáme, že ich už nepotrebujeme, keďže do značnej miery ovplyvňujú výkon (spomaľujú) a vyčerpávajú prostriedky operačného systému. Používanie systémových hákov správ musíme robiť s rozvahou, pretože hneď po aktivácii hákov si každá aplikácia dostávajúca niektorú so správ o ktorú máme záujem, automaticky načíta našu knižnicu a ponecháva si ju vo svojom pamäťovom priestore až do chvíle, kým nezastavíme hákovanie odhlásením. Každá procedúra ľubovoľného háku dostáva ako jeden zo svojich parametrov takzvaný *nCode*, ktorý určuje, či procedúra háku môže zachytenú správu spracovať (v prípade nezápornej hodnoty), alebo či musí procedúra vrátiť hodnotu získanú volaním funkcie *CallNextHookEx*. Toto

volanie spôsobí, že správa sa odovzdá na spracovanie hákovacej procedúre, ktorá je ďalšia v poradí v reťazci inštalovaných hákov operačného systému.



Obrázok č. 10 Reťazec hákov.

Mechanizmus hákovania operačného systému je často krát využívaný viacerými aplikáciami zároveň. Takáto situácia sa rieši vytvorením reťazca hákov, ktorým hákovaná správa prechádza (všetkými procedúrami hákov), samozrejme iba do chvíle, kým nie je niektorým z hákov v reťazci priechod správy zastavený, resp. zachytený. To však ešte stále nemusí znamenať, že zachytená správa sa nedostaví pôvodnému adresátovi. Aby zastaveniu správy v rámci reťazca hákov nedošlo, je nutné správne volať funkciu *CallNextHookEx*, ktorá odovzdá zachytenú správu ako parameter procedúry ďalšieho systémového háku.

2.4 Implementácia systémových hákov správ

Prvým krokom implementácie systémových hákov správ môže byť výroba dynamicky linkovanej knižnice, kde sa uplatňujú rutinné postupy ako výroba vstupného bodu vykonávania kódu modulu (*DllMain*), reakcia na dôvod pripojenia dynamickej knižnice (*DLL_PROCESS_ATTACH*), ktorý dostane ako parameter a samozrejme exportovanie všetkých potrebných funkcií na komunikáciu knižnice s aplikáciou, ktorá ju využíva. Po týchto krokoch je knižnica pripravená na implementáciu systémových hákov správ.

Základnou myšlienkou odchytávania správ pomocou systémových hákov je posielanie takzvaných skopírovaných správ prostredníctvom klasickej funkcie *SendMessage* aplikácii, ktorá je uvedená ako „adresát správy“, teda naša aplikácia, v ktorej chceme zachytené a skopírované správy prijímať. Na to však potrebujeme, ako aj parametre funkcie *SendMessage* napovedajú, handle na okno našej aplikácie

(„adresáta“). Handle našej aplikácie zisťujeme najlepšie práve v momente načítania našej knižnice, reakciou v *DllMain* na dôvod pripojenia *DLL_PROCESS_ATTACH*, kedy je vhodné funkciou *FindWindow* vyhľadať okno našej aplikácie a tým pádom získať aj jej handle.

Skopírovanú správu, teda štruktúru *COPYDATASTRUCT* nesúcu pôvodnú zachytenú správu, posielame našej aplikácii ako parameter správy *WM_COPYDATA*, ktorá je predovšetkým určená k posielaniu dát medzi aplikáciami. Umožňuje zasielať totiž ľubovoľne rozsiahle dátové štruktúry, ku ktorým sa na oboch stranách (vo vysielacej i prijímacej aplikácii) pristupuje cez ukazateľ. Na strane našej aplikácie (prijímacej) musíme v procedúre príslušného okna túto správu reagovať a patrične ju spracovávať. Aby mechanizmus našich systémových hákov „ožil“, musíme každý jeden hák (každú jeho procedúru) pred použitím zaregistrovať funkciou *SetWindowsHookEx*. Naopak, aby bola činnosť hákov riadne ukončená, tak pred ukončením našej aplikácie musíme zavolať so správnymi parametrami funkciu *UnhookWondowsHookEx*. V tomto momente sme už schopný zachytávať správy určené iným oknám, avšak aby boli háky správ absolútne odladené, neraz bude musieť aj skúsený programátor sedieť dlhé hodiny pred monitorom svojho počítača.

Záťaž systému je pri efektívne napísaných procedúr hákov na priemernom počítači minimálna. Háky ktoré zachytávajú správy týkajúce sa výnimočných udalostí (teda vyskytujú sa pomerne zriedka) až takú záťaž pre operačný systém nepredstavujú (*WH_SHELL* a zachytávanie udalosti vytvorenia nového top level okna), naopak háky zachytávajúce udalosť akou je pohyb myši, sa vyskytujú extrémne často a procedúram hákov tohto typu musíme venovať zvýšenú pozornosť.

2.5 Niektoré typy hákov

- WH_CALLWNDPROC
- WH_CALLWNDPROCRET
- WH_GETMESSAGE
- WH_SHELL
- WH_MOUSE
- WH_KEYBOARD_LL

WH_CALLWNDPROC

Procedúra tohto háku sa zavolá vždy, keď je poslaná správa niektorej z aplikácií pomocou funkcie *SendMessage*. Správu najskôr spracujú procedúry reťazca hákov a až potom je originálna správa doručená pôvodnej procedúre okna. Obmedzenie na hák tohto typu je, že nesmie zachytenú správu pred zavolaním funkcie *CallNextHookEx* nijako modifikovať, ani zadržať.

WH_CALLWNDPROCRET

Ako už názov typu sám vypovedá, je veľmi podobný háku typu *WH_CALLWNDPROC*. Podstatný rozdiel je však v dobe, kedy sú procedúry hákov volané. Kým procedúra háku typu *WH_CALLWNDPROC* je volaná ešte pred tým, než ju aplikácia prijme, procedúra háku typu *WH_CALLWNDPROCRET* sa zavolá až potom, čo príslušná procedúra okna spracuje správu zaslanú funkciou *SendMessage*. Rozdielne sú aj parametre procedúr. Procedúra háku typu *WHCALLWNDPROCRET* dostáva ako *lParam*, štruktúru pochopiteľne nesúcu informáciu aj o návratovej hodnote procedúry okna, ktoré je spracovalo.

WH_GETMESSAGE

Procedúra tohto háku je zavolaná vždy, keď je pomocou funkcie *GetMessage* alebo *PeekMessage* sa vyberie správa z fronty správ aplikácie. Pre parameter *nCode* platia rovnaké pravidlá ako pre vyššie uvedené typy, avšak tentoraz by mala procedúra háku zavolať funkciu *CallNextHookEx* aj v prípade, že hodnota *nCode* je nezáporná. V opačnom prípade by sa mohlo stať, že správa nebude pokračovať priechodom cez reťazec hákov, ale že ju hák zachytí. Dokonca je možné, že aplikácia, pre ktorú bola správa pôvodne určená, ju nedostane.

WH_SHELL

Procedúra háku tohto typu je zavolaná systémom vždy, keď nastane niektorá zo systémových udalostí, najmä zmeny aktívneho okna. Parameter *nCode* má v tomto prípade sémanticky iný význam. Tentoraz určuje typ udalosti, ktorá v systéme nastala a na ktorú naša procedúra reaguje. Týmito udalosťami môžu byť:

- HSHELL_WINDOWACTIVATED - aktívne top level okno sa zmenilo. Táto udalosť typicky nastane, keď užívateľ prechádza do okna inej aplikácie.
- HSHELL_WINDOWCREATED – v systéme sa vytvorilo nové top level okno. Táto udalosť typicky nastáva, keď užívateľ spustí novú aplikáciu.
- HSHELL_WINDOWDESTROYED – niektoré z top level okien bolo zrušené. Táto udalosť typicky nastáva, vo chvíli keď užívateľ zruší hlavné okno aplikácie.
- HSHELL_LANGUAGE – udalosť nastáva z pravidla vo chvíli, keď užívateľ prepne rozloženie klávesnice.
- HSHELL_TASKMAN – užívateľ aktivoval okno TaskManager (zoznam úloh)

WH_MOUSE

Windows volá hák typu *WH_MOUSE*, vždy vo chvíli, keď je pomocou funkcií *GetMessage* alebo *PeakMessage*, vybratá správa nesúca nejaké informácie o myši (o jej polohe, kliknutí tlačítok...). Je možné hákom tohto typu správu úplne zachytiť, teda nezasielať ju ďalej ani v reťazci inštalovaných hákov, ani aplikácii pre ktorú bola určená.

WH_KEYBOARD_LL

Hák typu *WH_KEYBOARD_LL* umožňuje zachytávať správy týkajúce sa klávesnice (stlačenie, uvoľnenie klávesy...) ešte pred tým, než sú zaradené do fronty správ aplikácie, alebo vlákna. Tento hák zachytáva aj správy ktoré boli umelo vygenerované funkciou *keybd_event*.

Existujú aj ďalšie typy hákov, ktoré sú schopné debugovať iné inštalované háky správ, konkrétne hák typu *WH_DEBUG* je volaný vždy pred tým ako OS zavolá ľubovoľný iný hák z reťazca hákov. Háky typu *WH_JOURNALRECORD* a *WH_JOURNALPLAYBACK* majú špeciálnu úlohu pri zaznamenávaní a prehrávaní vstupov užívateľa.

2.6 Nahrávanie užívateľského vstupu

Súčasť funkčnosti aplikácie Mouse Gestures je aj nahrávanie užívateľského klávesového vstupu. Je to jedna zo schopností aplikácie, ktorá pri správnom zaobchádzaní pomôže pri jej interpretácii (pri interpretácii nahratej nahrávky) ovládať aplikácie a ušetrí užívateľovi veľa času. V ďalšej podkapitole je presne popísaný systémový hák a dôvod prečo sa hák využíva, ktorý aplikácia Mouse Gestures používa na zachytávanie vstupu.

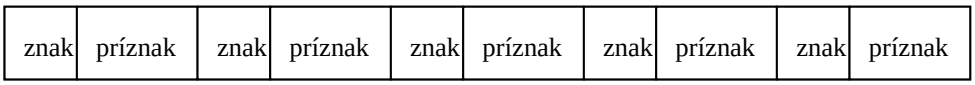
Ak sa teda nezaoberáme tým, ako zachytíme klávesové udalosti, tak sa zamerajme na to, v akom tvare a kam ich uložíme, aby boli neskôr pri vykonávaní významu gesta použiteľné a hlavne správne.

Požiadavky na dátové štruktúry

Keďže zachytením klávesového vstupu získame kód virtuálnej klávesy, čo je číslo z intervalu <1, 254> na uchovanie tejto informácie stačí dátový typ „char“. S týmto číslom pri hákovaní je ľahko zistiteľní aj príznak či bola klávesa stlačená alebo uvoľnená. Toto sú všetky informácie, ktoré je nutné uchovávať. Kvôli implementácii aplikácie Mouse Gestures je toto miesto v reťazci typu char, ktorý sa vyskytuje v štruktúre významu gesta.

Ukladanie klávesovej nahrávky do reťazca v Mouse Gestures

Reťazce opäť kvôli implementácii Mouse Gestures môžu byť dlhé maximálne 260 znakov, čo zodpovedá konštante MAX_PATH deklarovanej hlavičkovom súbore „Windows.h“. Čísla virtuálnych kláves spolu s príznakom stlačenia či uvoľnenia, sa efektívne a hlavne jednoducho uložia v reťazci za sebou. Ak je klávesová nahrávka dlhšia, ako 260 vstup sa rozdeľuje na niekoľko reťazcov nahrávky. Príznak stlačenia alebo uvoľnenia klávesy sa zapisuje bezprostredne za číslom virtuálnej klávesy (v prípade Mouse Gestures je to 1 pre stlačenie a 2 pre uvoľnenie, 0 je nepoužiteľná, keďže je znakom pre ukončenie reťazca).



Obrázok č. 11 Sekvenčné uloženie klávesovej nahrávky v reťazci.

2.7 Systémové háky a aplikácia Mouse Gestures

Aplikácia Mouse Gestures k svojej správnej funkčnosti implementuje zo systémových hákov tieto typy:

- WH_MOUSE_LL
- WH_KEYBOARD_LL
- WH_SHELL

Hák typu *WH_MOUSE_LL* implementuje, lebo musí mať po celý čas prehľad o aktuálnej pozícii myši a to v akejkoľvek situácii, ľubovoľnom mieste na monitore alebo kontexte. Nízko-úrovňový typ háku zabezpečuje zachytávanie aj tých správ, ktoré nie je možné s hákom typu *WH_MOUSE* zachytiť. Väčšinou sú takýmito správam generované pohybom myši v systémovom menu, alebo v ponuke „Štart".

Nevyhnutné bolo aj nasadenie nízko-úrovňového háku klávesnice typu *WH_KEYBOARD_LL* aby aplikácia bola schopná zachytávať aj kombinácie systémových kláves, teda klávesa „alt" v kombinácii s nejakou inou alebo podobné klávesy negenerujúce znak použiteľný v texte.

Systémový hák typu *WH_SHELL* pomáha aplikácii Mouse Gestures udržiavať stále aktualizovaný zoznam aktívnych (spustených aplikácií). Aby bola aplikácia schopná zistiť, ktoré má práve fokus (teda je aktívne, čo väčšinou znamená, že užívateľ s ním pracuje), reaguje na udalosť *HSHELL_WINOWACTIVATED*. Táto schopnosť vytvára podmienky pre definovanie množiny aplikácií pri ktorých nechceme, aby mala Mouse Gestures aktívny hák myši a tým sa v podstate stala neaktívnou. Naopak po zmene fokusu na okno, v ktorom chceme aby bola aktívna, automaticky inštaluje háky správ myši späť. Samozrejme užívateľ môže kedykoľvek aj manuálne deaktivovať aplikáciu (jej mechanizmus hákovania myši).

Všetky háky, ktoré Mouse Gestures využíva sa nachádzajú v dynamicky linkovanej knižnici „*Mouse Gestures Hooks.dll*". Všetky procedúry hákov sú napísané tak, aby mali čo najmenšiu záťaž na systém. Všetky reakcie na zachytené správy sú sústredené v spustiteľnej časti aplikácii (*Mouse Gestures.exe*), a teda modul s hákmi iba zasiela zachytené správy, čím sa náklady na prostriedky operačného systému na mechanizmus hákovania minimalizujú.

Kapitola 3

Vykonávanie významu gesta

3.1 Motivácia a popis problému

V štádiu keď dokážeme spoľahlivo rozpoznať gesto myši nakreslené na monitore nášho počítača, ktoré môže byť vďaka systémovým hákom (konkrétne vďaka háku myši) nakreslené kdekoľvek a kedykoľvek, zostáva stále nezodpovedanou otázkou, ako nášmu gestu priradiť nejaký význam, či ako už nejakým spôsobom nadefinovaný význam vykonať. Pri riešení tohto problému nestačí, že dokážeme zachytávať a prijímať správy od iných aplikácií, ale naopak, musíme vedieť nejaké správy ostatným aplikáciám aj posielať.

Aby sme boli schopný klasickým spôsobom (funkciou *SendMessage*) posielať správy iným aplikáciám potrebujeme rozhodne poznať 32 bitové číslo ktoré jednoznačne identifikuje ľubovoľný objekt systému nazývaný *handle*. Každé okno musí mať v operačnom systéme Windows takéto jednoznačné číslo, na ktoré sa systém odkazuje v prípade, že mu potrebuje zaslať nejakú správu. V systéme ďalej existuje v určitom okamžiku množstvo grafických objektov, akými sú ikony, kurzory, otvorené súbory atď. Tieto objekty majú tiež každý svoj vlastný jedinečný identifikátor, ktorým je práve handle.

Je prirodzené, že v ďalšej podkapitole sa budeme snažiť získať handle na okná aplikácií, aby bolo zasielanie správ pomocou funkcie *SendMessage* korektné. Ak teda už vlastníme handle na okno inej aplikácie, máme za sebou zhruba len polovicu cesty k správnemu a opakovanému vykonávaniu definovaného gesta.

Kvôli skutočnosti, že handle sú systémom prideľované podľa toho, ktorý je práve voľný, sa v žiadnom prípade nedá predpokladať, že handle na okno aplikácie ktorý už raz vlastníme, bude po zatvorený a otvorení okna, prípadne zatvorení a otvorení aplikácie, ktorej okno patrí, stále handle na naše pôvodné okno. Preto je nutné zaviesť aj iný mechanizmus, ktorým by sme dokázali vždy jednoznačne identifikovať aplikáciu a získať handle na jej ľubovoľné okno (prípadne objekt), alebo aspoň s istotou spoznať, že vyžadovaná aplikácia nie je spustená.

Ťažkou úlohou je aj zasielanie klávesových udalostí, ktorú budem v podkapitole „Simulácia klávesových udalostí“ riešiť simuláciou užívateľského vstupu (konkrétne toho, ktorý si užívateľ „nahral“ vďaka háku klávesnice).

Nasledujúce časti textu tejto práce môžu miestami pripomínať programátorskú dokumentáciou aplikácie Mouse Gestures, čo je spôsobené faktom, že prezentovaný mechanizmus, je jednak využívaný v Mouse Gestures a jednak je často krát jediný, ktorý poskytuje všetky potrebné možnosti.

3.2 Získavanie handle-ov spustených aplikácií

Ako býva zvykom, čím je problém väčší, tým väčšie množstvo riešení naň existuje. Výnimkou nie je ani problematika získania handle-ov iných okien v systéme. Uvediem niekoľko spôsobov.

Získanie zoznamu všetkých samostatných okien v systéme

Funkcia *EnumWindows* je jedným z najhorúcejších kandidátov na získanie potrebného handle. Táto funkcia nám dokonca poskytne handle hneď na všetky top level oná v systéme. Prvý parameter tejto funkcie je ukazateľ na procedúru špeciálneho tvaru, ktorá sa zavolá vždy keď je v systéme nájdené nejaké top level okno. Ukážka kódu, ktorý je schopní získať všetky handle top level okien v systéme:

```
BOOL CALLBACK EnumWindowsProc(HWND hWnd, LAPRAM lParam) {
    //v parametri hWnd mame k dispozicii aktualne najdeny handle
    return TRUE;
}
void findAllHandles() {
    EnumWindows(EnumWindowsProc, NULL);
}
```

Tento postup sa využíva aplikáciou Mouse Gestures pri inicializácii zoznamu aktívnych (bežiacich) aplikácií, ale aj pri zobrazovaní aktívnych top level okien v dialógovom okne „*New Gesture*“ pri definovaní významu gesta.

Nájdenie okna konkrétnej aplikácie

Funkcia *FindWindow* nájde požadované okno, ktoré ako prvé spĺňa zadané podmienky. Prvým parametrom je názov triedy hľadaného okna, druhým je text

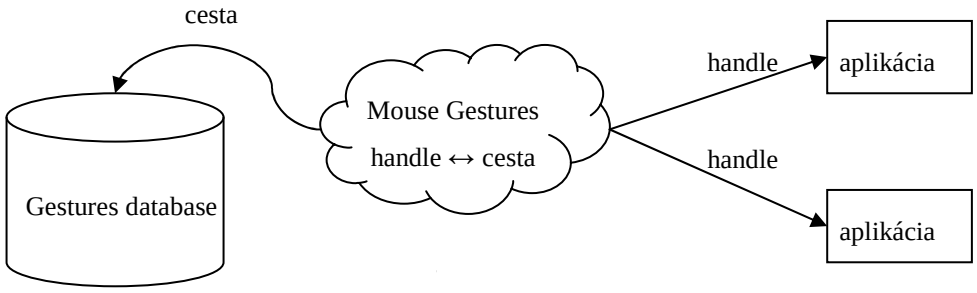
(titulok) okna. Ak neuvedieme niektorý z parametrov (teda použijeme hodnotu NULL), funkcia hľadá okná ktoré vyhovujú iba jednej podmienke. Ukážka získania handle na aplikáciu „Notepad“:

```
HWND handle = FindWindow("Notepad", NULL);
```

Ak žiadne z okien podmienke nevyhovuje, funkcia vracia NULL. V aplikácii Mouse Gestures je táto funkcia použitá v knižnici s hákmi, keď samotný modul háku vyhľadáva „miesto“, kam zachytené správy bude posielať, teda aplikáciu Mouse Gestures.

3.3 Správa aplikácií pomocou ciest k nim

Na to aby bol význam gesta myši vždy zachovaný (aj po vypnutí Mouse Gestures) bol v aplikácii Mouse Gestures vyvinutý mechanizmus, ktorý pacuje s cestami k aplikáciám a handle na jej okná využíva len pri zasielaní správ. Keďže po spustení aplikácie pridelí operačný systém jej objektom (a teda aj oknám) práve nevyužité handle a na to, že im pridelí zakaždým rovnaký sa nemôže žiaden racionálne pracujúci program spoľahnúť, zostáva jediným konštantným príznakom slúžiacim na spoľahlivú identifikáciu aplikácie len cesta k nej. Obrázok č. 11 demonštruje konverziu Mouse Gestures medzi cestou a handle-om aplikácie. Ako je z neho vidieť, práve cesty k aplikáciám sa ukladajú do vnútorných štruktúr, ale aj do binárnych súborov na HDD, aby nemusel byť význam gesta definovaný vždy, keď sa zmení handle príslušného okna.



Obrázok č. 12 Konverzia ciest a handle-ov v aplikácii Mouse Gestures

Zostáva už len vyriešiť otázku konverzie ciest k aplikáciám na ich handle, prípadne handle aplikácií na cesty k nim. Opäť popíšem niekoľko spôsobov ako je možné tieto konverzie realizovať.

Získanie plného mena súboru aplikácie

V mnohých príručkách pre programovanie vo Windows, som našiel nasledujúcu avšak nie vždy fungujúcu variantu získania cesty (teda plného mena súboru aplikácie). Problém spočíva v tom, že handle ktorý chceme konvertovať môže patriť síce top level oknu aplikácie avšak jeho procedúra môže byť umiestnená v nejakom module, alebo v dynamicky linkovanej knižnici a nie priamo v spustiteľnom súbore aplikácie. Tým pádom v niektorých prípadoch nedostaneme cestu k spustiteľnému súboru, ale cestu k tomuto modulu. Funkcia teda pracuje správne, ale nie vždy tak, ako by užívateľ mohol očakávať.

```
void GetPathFromHandle(HWND hWnd, TCHAR* path) {
    ZeroMemory(path, sizeof(TCHAR) * MAX_PATH);
    GetWindowModuleFileName(hWnd, path, MAX_PATH);
}
```

Správne získanie cesty k aplikácii z jej handle-u

Nasledujúci fragment kódu (funkcia) použitý priamo v aplikácii Mouse Gestures demonštruje konverziu handle-u (dostane ho ako parameter) na cestu (vráti ju ako výstupný parameter). Takže tento krát správna varianta kódu vyzerá nasledovne:

```
int GetPathFromHandle(HWND& hWnd, TCHAR* path) {

    ZeroMemory(path, sizeof(TCHAR) * MAX_PATH);
    DWORD dwProcessID;
    GetWindowThreadProcessId(hWnd, &dwProcessID);
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
                      PROCESS_VM_READ, FALSE, dwProcessID);
    HMODULE hModule = (HMODULE)(GetClassLongPtr(hWnd,
                                GCLP_HMODULE));

    return GetModuleFileNameEx(hProcess, hModule, path, MAX_PATH);
}
```

Kód je síce zložitejší, ale rozhodne vždy vráti cestu k spustiteľnému súboru a nie k jednému z modulov aplikácie. Tým, že získa handle procesu ktorému daný handle

okna patrí, potom ho následne kombinuje s príslušným handle na modul vo funkcii *GetModuleFileNameEx*.

Konverzia cesty na handle aplikácie

Konverzia cesty na handle aplikácie prebieha pomocou dátových štruktúr aplikácie Mouse Gestures. Konkrétne si udržiava mapu dvojíc <cesta k aplikácii, aktuálny handle>. Nie je teda problém vyhľadať handle na ľubovoľnú bežiacu aplikáciu v logaritmickom čase vzhľadom na počet spustených (evidovaných) aplikácií, keďže štruktúra „map“ z knižnice STL je implementovaná ako červeno čierny strom. Viac o stromových štruktúrach je možné nájsť v publikácii P. Tö pfera [2] na strane 74. Aktuálnosť zoznamu spustených aplikácií, ich ciest a handle-ov udržuje Mouse Gestures vďaka inštalovanému háku typu *WH_SHELL* v operačnom systéme a príslušnou reakciou na každú udalosť typu *HSHELL_WINDOWCREATED* a *HSELL_WINDOWDESTROYED*.

3.4 Simulácia klávesových udalostí

Ak užívateľ nahrá svoj klávesový vstup pomocou aplikácie Mouse Gestures a jej inštalovaného háku, očakáva, že si ho niekedy v blízkej, či vzdialenej budúcnosti bude môcť opäť vďaka Mouse Gestures prehrať. Keďže nie je v moci žiadneho softwaru fyzicky stláčať a uvoľňovať klávesy na užívateľovej klávesnici (aj keď niekedy by bolo možno ľahšie naprogramovať práve toto), musí užívateľský vstup z klávesnice simulovať.

Možnosti simulácie

V zásade existujú len dve, pričom jedna je nadstavbou (ak sa to tak dá označiť) druhej. Presnejšie povedané jedna využíva služieb druhej. Konkrétne je reč o dvoch funkciách na simulovanie klávesového vstupu: *SendInput* a *keybd_event*.

Funkcia *SendInput* využíva na simulovanie klávesového vstupu spomínanú *keybd_event*. Avšak medzi týmito funkciami je zásadný rozdiel. Zatiaľ čo funkcia *SendInput* je užívateľsky príjemnejšia (ako parameter preberá reťazec klávesových vstupov), stará sa ešte aj o to, aby ňou simulované klávesové udalosti neboli prerušené žiadnou inou klávesovou udalosťou.

Funkcia keybd_event je umiestnená na nižšej (užívateľskej) úrovni, avšak okrem simulácie stlačenia alebo uvoľnenia jednej klávesy (ktorú dostane ako parameter) nedokáže nič iné. A práve preto našla v aplikácii Mouse Gestures svoje uplatnenie, keďže je v nej povolené zasahovať počas simulácie vstupu iným klávesovým (či už simulovaným alebo nie) vstupom. Jedným z parametrov tejto funkcie je takzvaný virtuálny kód klávesy, teda číslo z intervalu <1, 254>. Iným parametrom je príznak špecifikujúci či sa má jednať o stlačenie, resp. uvoľnenie klávesy (*KEYEVENTF_EXTENDEDKEY* resp. *KEYEVENTF_KEYUP*).

3.5 Implementácia štruktúr nesúcich význam gesta

V tejto podkapitole bude rozobraté, aké dátové štruktúry a základné vzťahy medzi nimi, implementuje aplikácia Mouse Gestures, aby bola schopná význam gesta nielen uchovať, ale neskôr aj vykonávať (interpretovať). Pod pojmom „exekúcia gesta“ sa v ďalšom texte bude rozumieť doba, pokiaľ je rozpoznané gesto vo fáze vykonávania jeho významu.

Ak by som zhrnul naše doterajšie poznatky z tejto kapitoly (dokážeme nájsť handle na ľubovoľnú spustenú aplikáciu, sme schopný naopak z ľubovoľného handle získať cestu k jeho spustiteľnému modulu, vieme ako simulovať klávesový vstup) zistíme, že práve implementácia štruktúr nesúcich význam gesta je tou poslednou vecou, ktorá môže chýbať niekomu, kto v tejto práci hľadá návod, ako aplikáciu podobnú Mouse Gestures naprogramuje sám.

Štruktúry nesúce význam gesta sú totiž kľúčovou záležitosťou každej podobnej aplikácie. Musia byť dostatočne rýchle a efektívne, nesmú uchovávať redundantné informácie, musia byť schopné zápisu na HDD v podobe binárnych dát a samozrejme musia byť prehľadné aby sa v nich aj sám užívateľ aplikácie ľahko orientoval. Nehodlám uvádzať podrobné členenie kódu na konkrétne triedy v Mouse Gestures (tieto informácie sú dostupné v programátorskej dokumentácii), ale pokúsim sa vysvetliť ako sú približne navrhnuté prečo práve tak.

Štruktúra gesta

Každé gesto nesie informácie o reťazci gesta ktorý ho „spúšťa“, o svojom význame, prípadne o svojom pomenovaní. V prípade štruktúry, ktorá predstavuje

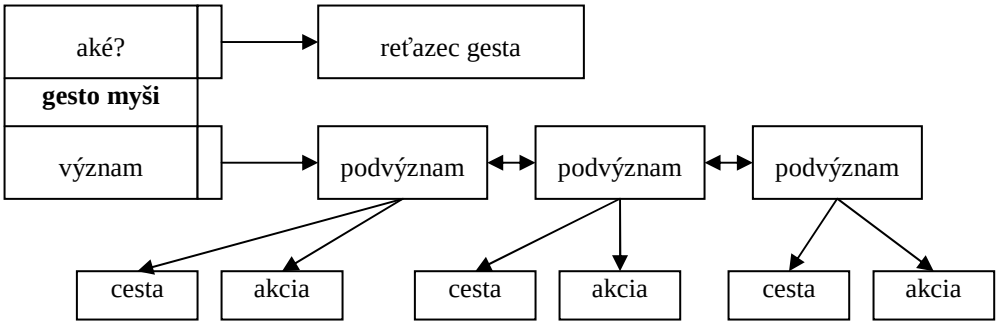
reťazec gesta je jednoznačné, že najvhodnejšie na jeho reprezentáciu je použiť pole, alebo klasický reťazec. Obe varianty však musia byť schopné dynamicky sa rozširovať, keďže reťazec gesta môže byť teoreticky bez obmedzenia. Čo sa týka štruktúry pre reprezentáciu pomenovania, nechám na uvážení každého (reťazec:).

Štruktúra významu gesta

Najzaujímavejšou štruktúrou obsiahnutou v štruktúre gesta je teda jeho samotný význam. Keďže jeden význam gesta myši môže pracovať s viacerými aplikáciami, so simuláciou užívateľského vstupu, ale zároveň môže posielať aj správy konkrétnym aplikáciám (napr.: príkaz na minimalizáciu jej top level okna), je nutné aby bol štruktúrovaný a skladal sa z menších podvýznamov. V aplikácii Mouse Gestures je implementovaný ako vektor menších celkov, každý nesúci jednu informáciu o význame.

Definícia: Nazvime najmenšiu časť významu gesta nesúcu nedeliteľnú informáciu ako *podvýznam*.

Z podvýznamu gesta musí byť zrejmé, čo sa má vykonať a hlavne na akej aplikácii. Prirodzene teda obsahuje cestu ku konkrétnej aplikácii a nejaký príznak, o akú akciu ide.

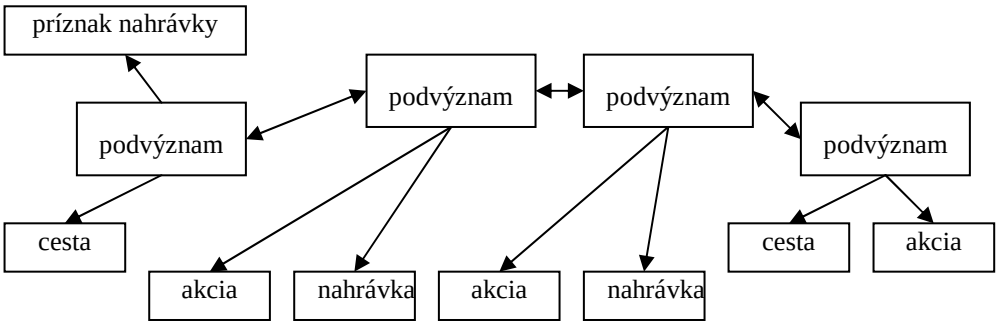


Obrázok č. 13 Štruktúra gesta.

Pri vykonávaní významu gesta sa teda sekvenčne prechádza zoznam jeho podvýznamov a vykonávajú sa jeho jednotlivé podvýznamy. Je zrejmé, že každý podvýznam môže byť definovaný vždy na inej aplikácii, teda obsahuje inú cestu. Tým sa dosiahne, že význam gesta môže pri exekúcii pracovať s viacerými aplikáciami. Nezodpovedanou otázkou stále zostáva ako a kde skladovať klávesovú nahrávku užívateľského vstupu. Keďže by bolo úplne zbytočné vytvárať ďalšie

priestory v podvýzname významu gesta na skladovanie klávesovej nahrávky, aplikácia Mouse Gestures v ňom využíva existujúci buffer, používajúci sa na ukladanie cesty k aplikácii. Ak chceme dosiahnuť aby sa štruktúra významu gesta nezmenila, musíme zaviesť do zoznamu jednotlivých podvýznamov novú logiku.

Užívateľské klávesové nahrávky jednoducho označíme špeciálnym príznakom akcie a cestu k aplikácii nad ktorou sa nahrávka má vykonávať vložíme pred sekvenciu podvýznamov s nahrávkou. To že hovoríme o sekvencii podvýznamov s klávesovou nahrávkou, je zapríčinené obmedzenou veľkosťou buffera (MAX_PATH), keďže je primárne určený len na udržiavanie cesty k aplikácii.



Obrázok č. 14 Štruktúra významu gesta obsahujúci klávesovú nahrávku.

Kapitola 4

Zostavenie gesta s Mouse Gestures

4.1 Motivácia a popis problému

Zostavenie funkčného a spoľahlivého gesta myši je často krát náročnou úlohou. V tejto kapitole budú demonštrované všetky úskalia a princípy tvorby efektívneho gesta myši pomocou Mouse Gestures. Čím komplexnejšie významy gest užívateľ vytvára, tým viac sa vystavuje riziku, že jedna z akcií podvýznamu neprebehne korektne (napr.: kvôli iným podmienkam pri exekúcii ako pri nahrávaní klávesového vstupu) a tým dramaticky ohrozí aj zvyšok exekúcie.

4.2 Výber aplikácií pre simuláciu vstupu

Spoľahlivou aplikáciou pre simuláciu užívateľského vstupu sa myslí taká aplikácia, ktorá medzi dvoma spusteniami nezmení podmienky pre vstup, to znamená, že sa jej ovládacie prvky reagujú vždy rovnako. Takouto aplikáciou je bezpochyby príkazový riadok operačného systému Windows, keďže žiadne ovládacie prvky (okrem konzoly) nemá. Užívateľ však musí pri vytváraní klávesovej nahrávky aj v tomto prípade predvídať všetky možné počiatočné stavy konzoly po jej spustení. Aplikácia Mouse Gestures len simuluje v daný okamžik klávesový vstup a teda nie je v jej silách, či akcie, ktoré vzniknú vďaka simuláciu sú adekvátne (t.j. či zodpovedajú užívateľovým predstavám). Napríklad exekúcia klávesovej nahrávky, ktorá simuluje pohyb a vyberá položky v menu, v ktorom sa pravidelne (nepredvídateľne) menia položky (prípadne len ich pozície), je vhodným kandidátom na vyvolanie nechcených udalostí.

Naopak gesto myši, ktoré preberá funkciu makroprocesora textu a počas exekúcie doplní určitý text v textovom poli okna, ktoré má práve fokus, je vhodné a určite aj bezpečné.

Rozhodne nie je bezpečné pripájať sa pomocou Mouse Gestures (naslepo vyplní citlivé prihlasovacie údaje) na konferencii počas projekcie plochy monitora na elektronický prístup k svojmu kontu.

Koniec koncov je na užívateľovi, aby odhalil úskalia každej aplikácie a časom určite vyprodukuje sadu efektívnych a hlavne bezpečných gest myši.

4.3 Optimálne kreslenie gesta

Ak už má užívateľ nadefinovaných niekoľko gest, chcel by určite dokázať tieto gestá vyvolať. A pri vyvolávaní (teda pri kreslení) gesta, chce aby bol jeho výtvor (gesto) rozpoznaný. Vtedy by si mohol položiť otázku akým štýlom kresliť gestá aby bola úspešnosť rozpoznania čo najvyššia. Otázka optimálneho kreslenia gest zostane však navždy nezodpovedanou. Nedá sa definovať ako vyzerá dobre, či zle nakreslené gesto a akým spôsobom sa to, či ono kreslí. Je faktom, že do zásadnej miery rozpoznávanie ovplyvňuje hustota bodov v trajektórii gesta, t.j. príliš pomalé, či rýchle kreslenie na úspešnosti rozhodne nepridáva. Užívateľ aplikácie Mouse Gestures má možnosť ovplyvniť parametre jednotlivých algoritmov, prípadne si sám vybrať algoritmus, ktorý čo najviac zodpovedá jeho schopnostiam a potrebám.

4.4 Ukážka zostavenia jednoduchého gesta

Ako ukážka zostavenia jednoduchého gesta môže byť použitá nasledujúca úloha: Po nakreslení, rozpoznaní a vykonaní významu gesta, chcem aby sa na ploche „objavil“ súbor „*hello world – pi.txt*“ obsahujúci text „Hello world!“ a v novom riadku pod týmto textom aby bola hodnota čísla pí. Je zrejmé, že táto úloha má nespočetné množstvo riešení a každé z nich vygeneruje požadovaný súbor. Predpoklad uvedeného riešenia je, že operačný systém Windows, na ktorom je aplikácia Mouse Gestures spustená, obsahuje základné aplikácie ako je poznámkový blok a kalkulačka.

Po nakreslení nového gesta a zobrazení dialogu *New Gesture* pokračujem napríklad krokmi:

- vyplním názov gesta: „hello world - pi“
- vyberiem cestu k aplikácii Notepad: C:\Windows\notepad.exe
- z ponuky vyberiem *Start* a stlačím *Add*

- stlačím *Record* a napíšem do okna Notepadu (ktorý sa medzičasom spustil) text: „Hello world!“ a nakoniec stlačím klávesu enter. Zastavím nahrávanie.
- vyberiem novú cestu k aplikácii: C:\Windows\system32\calc.exe
- z ponuky vyberiem *Start* a stlačím *Add*
- stlačím *Record* a postupne stláčam klávesy „p“, „ctrl + c“ a „alt + F4“. Zastavím nahrávanie.
- z ponuky aktívnych aplikácií vyberiem opäť Notepad a stlačím *Record*
- nasleduje klávesová sekvencia: „ctrl + v“, „ctrl + s“, „hello world – pi“, podľa potreby päť stláčame klávesy, aby sme sa dostali na správne miesto. Klávesom enter vytvorený súbor uložíme na požadované miesto. Príkazom „alt + F4“ ukončím Notepad. Opäť ukončím aj stav nahrávania.
- v dialogu *New Gesture* stlačím *Ok*
- nové gesto je nadefinované a pripravené k použitiu

Záver

Téma rozpoznávania a vykonávania gest, či už myši, alebo iných vstupných zariadení je zaujímavá, málo preskúmaná a ešte menej publikovaná. Preto dúfam, že moja záverečná práca obohatí túto chudobnú oblasť ako o teoretické poznatky tak aj o software v podobe priloženej aplikácie Mouse Gestures.

Mnohé postupy použité pri implementácii tohto programu a teda aj v práci sa dajú určite vymyslieť inak a možno šikovnejšie, či efektívnejšie. V žiadnom prípade si táto práca neosobuje právo ukážkového alebo najlepšieho postupu pri konštrukcii podobne zameranej aplikácie. Avšak poznatky a z nich prameniace dôsledky, ktoré táto práca zapuzdruje, môžu byť užitočné a nápomocné aj ďalším projektom v mnohých menej, či viac podobných oblastiach.

Použitá literatúra

[1] Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel: Moderní počítačová grafika, Computer Press, Praha, 2005

[2] Töpfer P.: Algoritmy a programovací techniky, Prometheus, Praha, 1995

[3] Radek Chalupa, 1001 tipů a triků pro Visual C++, Computer Press, Brno, 2003